

Tips for Shutdown and Process-Switching

2012/10/19

Version 1.1

**The content of this document is highly confidential
and should be handled accordingly.**

CONFIDENTIAL

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

1	Tips for Shutdown and Process-Switching	4
1.1	ProcUI Functions While the Application Is Running	4
1.2	Before Process-Switching	4
1.3	After Recovering From a Process Switch.....	5
1.4	Shutting Down the Application	5

1 Tips for Shutdown and Process-Switching

1.1 ProcUI Functions While the Application Is Running

Pass TRUE to ProcUIProcessMessages.

The `ProcUIProcessMessages` function can perform process-switching in blocking mode (when `TRUE` is passed as the argument) or in non-blocking mode, but blocking mode is easier to implement.

Pass `TRUE` to this function unless you have a special reason not to.

If you pass `TRUE` as the argument, you will only need to implement the following four functions used by ProcUI: `ProcUIInit`, `ProcUIDrawDoneRelease`, `ProcUIProcessMessages`, and `ProcUIShutdown`.

Call ProcUIProcessMessages immediately after GX2DrawDone.

There must be no instructions left on the GPU when you perform a process switch.

You must therefore call the `ProcUIProcessMessages` function immediately after calling the `GX2DrawDone` function.

Stop the thread where a GX2 command is issued when calling the ProcUIProcessMessages function.

After performing a process switch using the `ProcUIProcessMessages` function, threads other than the one that called `ProcUIProcessMessages` will overrun slightly. To avoid this problem, stop the thread where a GX2 command is issued when calling the `ProcUIProcessMessages` function.

For example, if you have an asynchronous thread for loading that also creates a display list, you must use a mutex or other exclusive locking construct to lock the place where the display list is created and the place where the `ProcUIProcessMessages` function is called.

Using ProcUISetBucketStorage and ProcUISetMEM1Storage:

You can restore the Foreground Bucket and MEM1 areas by using the `ProcUISetBucketStorage` and `ProcUISetMEM1Storage` functions, but these have disadvantages in terms of memory usage and process-switching time.

If you are concerned about memory usage or process-switching time, then Nintendo recommends not using these functions, and designing your application so that it is acceptable for the Foreground Bucket and MEM1 to be corrupted after a process switch.

1.2 Before Process-Switching

Call GX2SetTVEnable and GX2SetDRCEnable before process-switching.

In addition to clearing the black screen, these functions also set a flag that tells the system that the application has allowed rendering.

When a process releases the foreground, GX2 can store the last frame that the application displayed; this flag determines whether to store the frame.

The last frame will not be stored unless the application sets the flag by calling the `GX2SetTVEnable` or `GX2SetDRCEnable` function with `TRUE` as the argument.

In addition, under the current implementation, at each process switch this flag is cleared after the last frame is stored. Therefore, despite the fact that the black screen is cleared when the application gets the foreground, the application must pass `TRUE` to these functions every time.

The simplest implementation is to pass only `GX2_TRUE` and never `GX2_FALSE` into the `GX2SetTVEnable` and `GX2SetDRCEnable` functions, and to always call them with the `ProcUIDrawDoneRelease` function, before process-switching.

Notes for Mutex Locking When Moving to Background

When a user presses the HOME Button during gameplay, the game moves to the background, but some games continue to run in the background.

For example, an online game might communicate in the background in order to maintain its connection. Only core 2 is available in the background. Core 0 and core 1 are not available.

If the game goes into the background with a thread on core 0 or core 1, and the thread leaves a mutex locked, then that lock will not be released until the game goes back to the foreground.

If a thread on core 2 tries to lock that mutex, it will block.

1.3 After Recovering From a Process Switch

After recovering from a process switch, restore the application's context by calling `GX2SetContextState`.

When you return from a process switch, the render state will have been overwritten. You must therefore restore the application's context by calling the `GX2SetContextState` function.

Think of this function as being equivalent to the `GX2CopyColorBufferToScanBuffer` and `GX2ClearBuffersEx` functions.

1.4 Shutting Down the Application

Use the `_Exit` function if you do not want to call the destructors of C++ static objects.

You have three options for exiting your application: return from the `main` function, call the `exit` function, or call the `_Exit` function. Of these, returning from `main` and calling `exit` will both call the destructors of C++ static objects.

The order in which the destructors of static objects are called is undefined. As a result, objects with interdependencies could hang if you are not careful with your implementation.

It is difficult to resolve this issue cleanly if you code the application's exit processing late in development.

In this case, you can use the `_Exit` function to exit the application without calling the destructors of C++ static objects.

Call `OSBlockThreadsOnExit` when shutting down.

As with what happens after process-switching, threads other than the one that called the `exit` function will overrun slightly.

You can make the exit function wait until these other threads stop by calling the `OSBlockThreadsOnExit` function.

Call `AXQuit` when exiting.

Cafe-SDK 2.07.03 P5 requires you to call the `AXQuit` function when exiting the application.

(This problem is fixed in Cafe-SDK version 2.08.01.)

All company and product names in this document are the trademarks or registered trademarks of their respective companies.

© 2012 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed, or loaned in whole or in part without the prior approval of Nintendo.